24 | 全球化部署：如何打造近在咫尺且永不宕机的数据库?

2020-10-02 王磊

分布式数据库30讲 进入课程 >

讲述：王磊
时长 22:31 大小 20.62M

你好，我是王磊，你也可以叫我 Ivan。

这一讲我们要聊的是“全球化部署”，其实这个词在我们的课程中已经出现很多次了。我猜说不定你心里一直有个问号：“全球化啊，这么高大上的东西和我有关系吗？”耐心看完这一讲，我相信你会有新的理解。

我们不妨给全球化部署起一个更接地气的小名，“异地多活”。怎么样，感觉亲切多了吧？全球化部署本质就是全球范围下的异地多活。总体上看，异地多活的直接目标是要预防区域级的灾难事件，比如城市级的断电，或是地震、洪水等自然灾害。也就是说，在这些灾难发生时，要让系统还能保障关键业务的持续开展。

因此，这里的“异地”通常是指除同城机房外，在距离较远的城市配备的独立机房，在物理距离上跳出区域级灾难的覆盖范围。这个区域有多大呢？从银行业的实践来看，两地机房的布局通常会部署在南北或者东西两个大区，比如深圳到上海，或者北京到武汉，又或者北京到西安，距离一般会超过 1000 公里。

对于银行业的异地机房建设，监管机构是有具体要求的，也就是大中型银行的“两地三中心”布局。而对于互联网行业来说，虽然没有政策性要求，但业务本身的高可用需求，也推动了头部公司进行相应的布局。

说完了“异地”这个概念，我们再来看异地多活是怎么回事。首先，异地多活是高可用架构的一种实现方式，它是以整个应用系统为单位，一般来说会分为应用和数据库两部分。

应用部分通常是无状态的，这个无状态就是说应用处理每个请求时是不需要从本地加载上下文数据的。这样启动多个应用服务器就没有什么额外的成本，应用之间也没有上下文依赖，所以就很容易做到多活。

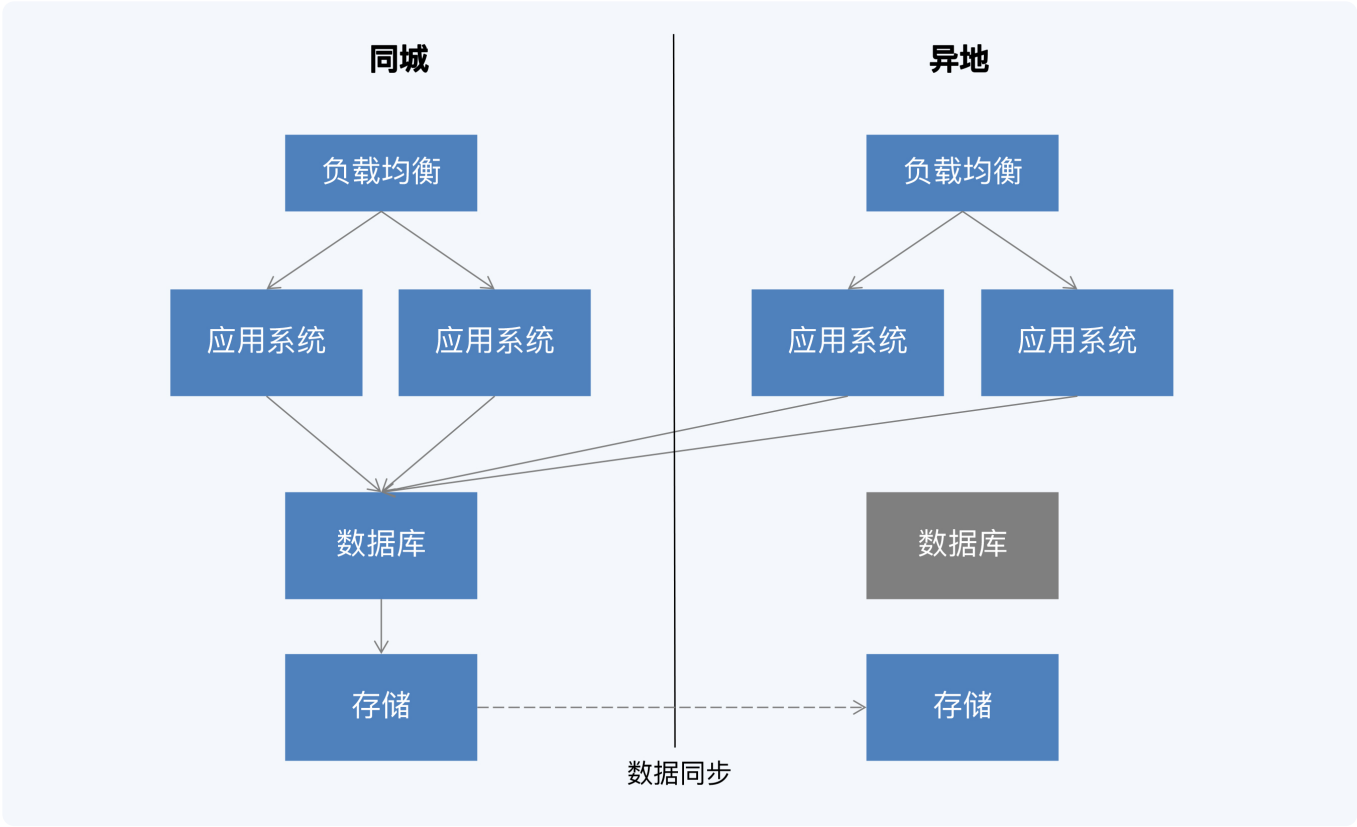
数据库节点要最终持久化数据，所有的服务都要基于已有的数据，并且这些数据内容还在不断地变化。任何新的服务节点在接入这个体系后，相互之间还会存在影响，所以数据库服务有逻辑很重的上下文。因此数据库的多活的难度就大多了，也就产生了不同版本的解读。

单体数据库

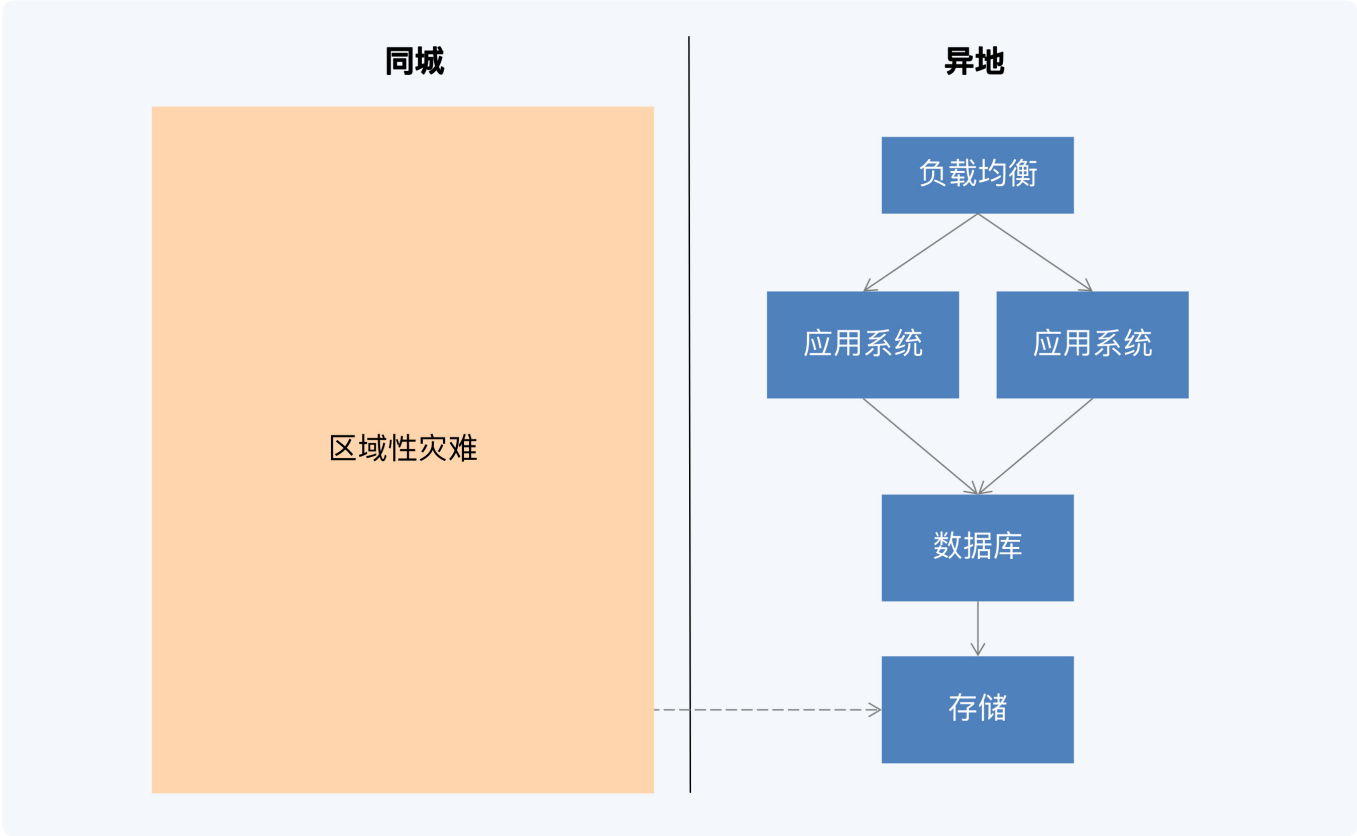
数据库层面的异地多活，本质上是要实现数据库的高可用和低延迟，也就是我们在标题中说“永不宕机”和“近在咫尺”。即便是单体数据库时代的技术方案，也是朝着这个方向努力的，我们不妨先来看看。

异地容灾

异地容灾是异地多活的低配版，它往往是这样的架构。



整个方案涉及同城和异地两个机房，都部署了同样的应用服务器和数据库，其中应用服务器都处于运行状态可以处理请求，也就是应用多活。只有同城机房的数据库处于运行状态，异地机房数据库并不启动，不过会通过底层存储设备向异地机房同步数据。然后，所有应用都链接到同城机房的数据库。



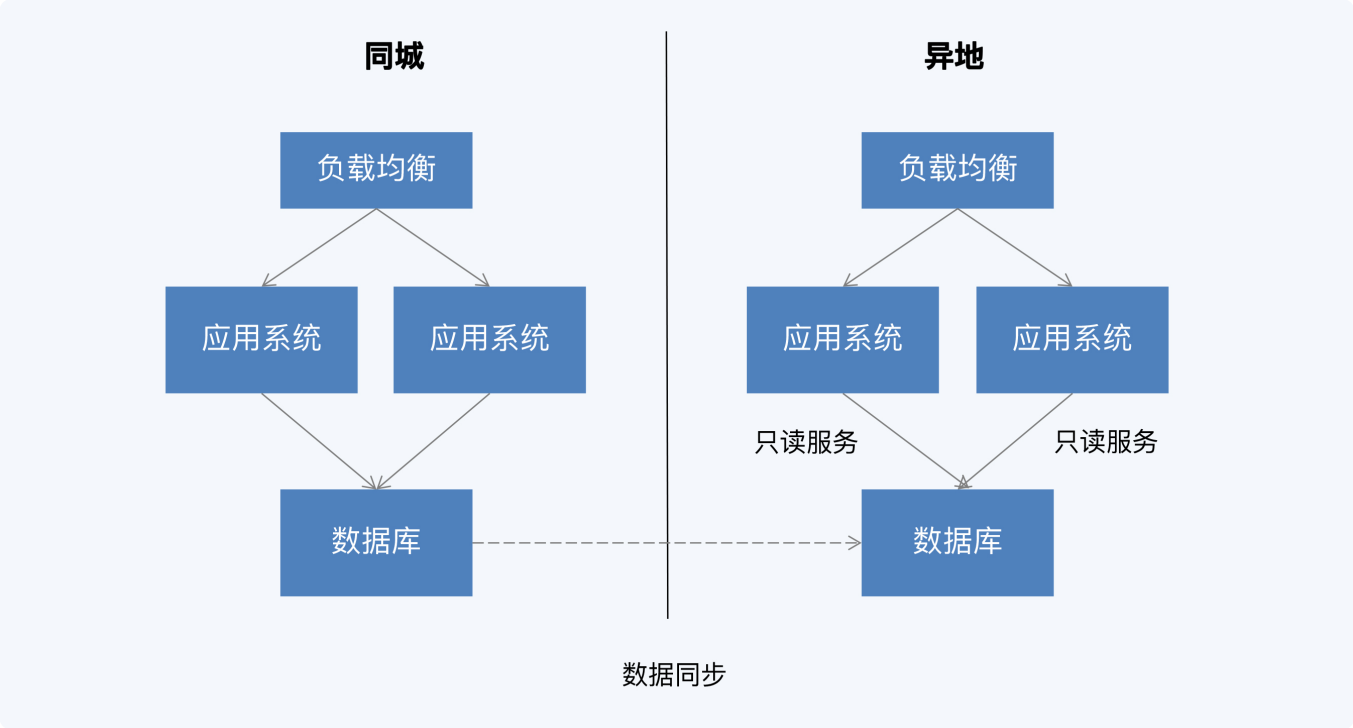
这样当同城机房整体不可用时，异地机房的数据库会被拉起并加载数据，同时异地机房的应用切换到异地机房的数据库。

显然，这个多活只是应用服务器的多活，两地的数据库并不同时提供服务。这种模式下，异地应用虽然靠近用户，但仍然要访问远端的数据库，对延迟的改善并不大。在系统正常运行情况下，异地数据库并没有实际产出，造成了资源的浪费。

按照正常的商业逻辑，当然不能容忍这种资源浪费，所以有了异地读写分离模式。

异地读写分离

在异地读写分离模式下，异地数据库和主机房数据库同时对外提供服务，但服务类型限制为只读服务，但只读服务的数据一致性是不保证的。



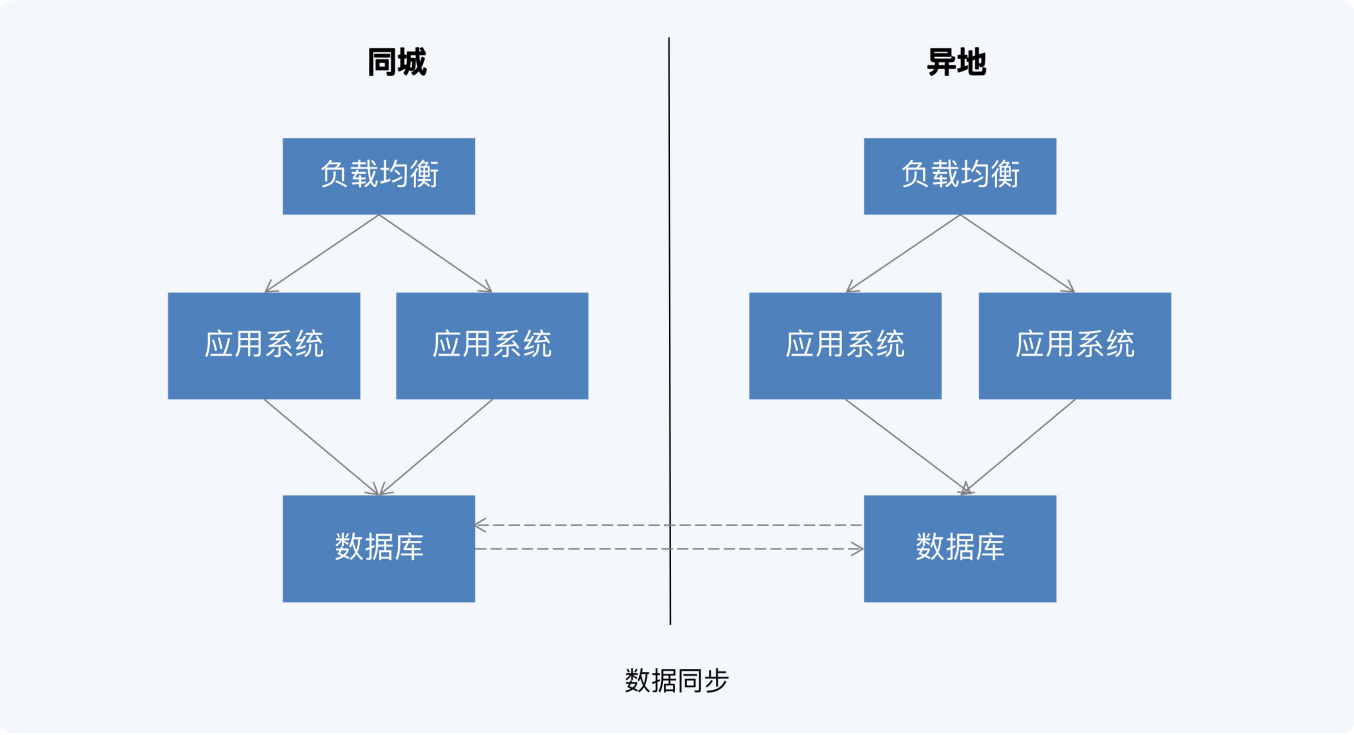
当主机房完全不可用时，异地机房的运作方式和异地容灾模式大体是一样的。

读写分离模式下，异地数据库也投入了使用，不再是闲置的资源。但是很多场景下，只读服务在业务服务中的占比还是比较低的，再加上不能保证数据的强一致性，适用范围又被进一步缩小。所以，对于部分业务场景，异地数据库节点可能还是运行在低负载水平下。

于是，我们又有了进一步的双向同步模式。

双向同步

双向同步模式下，同城和异地的数据库同时提供服务，并且是读写服务均开放。但有一个重要的约束，就是两地读写的对象必须是不同的数据，区分方式可以是不同的表或者表内的不同记录，这个约束是为了保证两地数据操作不会冲突。因为不需要处理跨区域的事务冲突，所以两地数据库之间就可以采用异步同步的方式。



这个模式下，两地处理的数据是泾渭分明的，所以实质上是两个独立的应用实例，或者可以说是两个独立的单元，也就是我们第 1 讲说的单元化架构。而两个单元之间又相互备份数据，有了这些数据可以容灾，也可以开放只读服务。当然，这个只读服务同样是不保证数据一致性的。

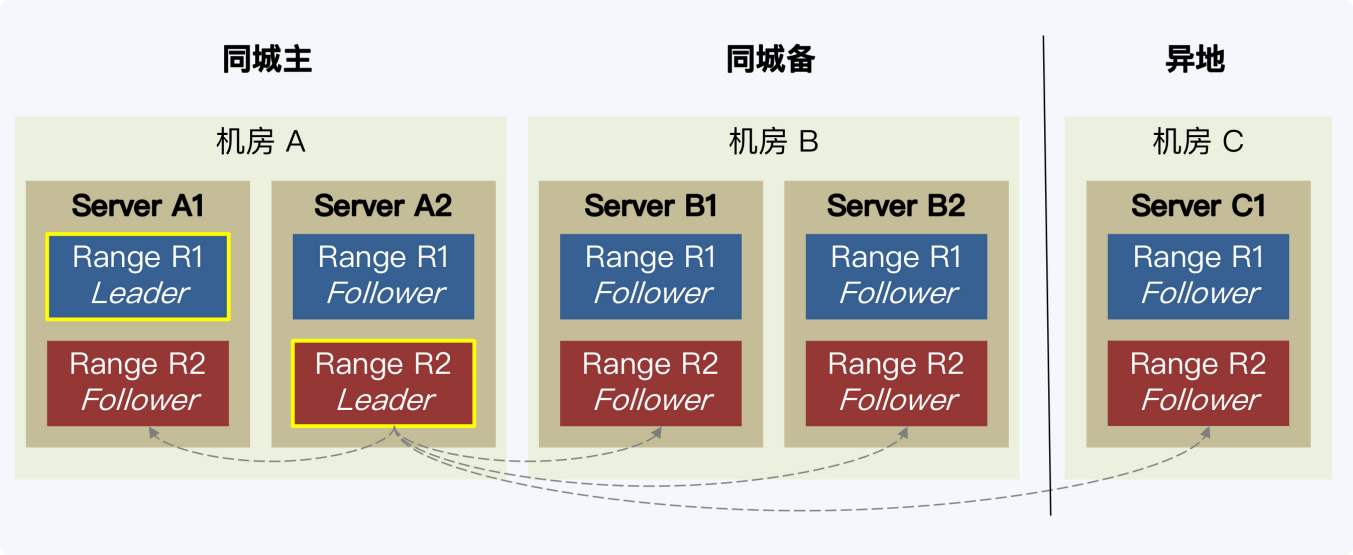
可以说，双向同步是单元化架构和异地读写分离的混合，异地机房的资源被充分使用了。但双向同步没有解决一个根本问题，就是两地仍然不能处理同样的数据，对于一个完整的系统来说，这还是有很大局限性的。

分布式数据库

分布式数据库的数据组织单位是更细粒度的分片，又有了 Raft 协议的加持，所以就有了更加灵活的模式。

机房级容灾（两地三中心五副本）

比较典型的分布式数据库部署模式是两地三中心五副本。这种模式下，每个分片都有 5 个副本，在同城的双机房各部署两个副本，异地机房部署一个副本。



这个模式有三个特点：

1. 异地备份

保留了“异地容灾”模式下的数据同步功能，但因为同样要保证低延迟，所以也做不到 RPO（Recovery Point Objective， 恢复点目标）为零。

2. 容灾能力

如果同城机房有一个不可用或者是同城机房间的网络出现故障，异地机房节点的投票就会发挥作用，依然可以和同城可用的那个机房共同达成多数投票，那么数据库的服务就仍然可以正常运行，当然这时提交过程必须要异地通讯，所以延迟会受到一定程度影响。

3. 同城访问低延迟

由于 Raft 或 Paxos 都是多数派协议，那么任何写操作时，同城的四个副本就能够超过半数完成提交，这样就不会因为与异地机房通讯时间长而推高数据库的操作延迟。

两地三中心虽然可以容灾，但对于异地机房来说 RPO 不为零，在更加苛刻的场景下，仍然受到挑战。这也就催生了三地五副本模式，来实现 RPO 为零的城市级容灾。

城市级容灾（三地五副本）

三地五副本模式是两地三中心模式的升级版。两个同城机房变成两个相临城市的机房，这样总共是在三个城市部署。

这种模式在容灾能力和延迟之间做了权衡，牺牲一点延迟，换来城市级别的容灾。比如，在北京和天津建立两座机房，两个城市距离不算太远，延迟在 10 毫秒以内，还是可以接受的。不过，距离较近的城市对区域性灾难的抵御能力不强，还不是真正意义上的异地。

顺着这思路，还有更大规模的三地五中心七副本。但无论如何，只要不放弃低延迟，真正异地机房就无法做到 RPO 为零。

无论是两地三中心五副本还是三地五副本，它们更像是单体数据库异地容灾的加强版。因为，其中的异地机房始终是一个替补的角色，而那些异地的应用服务器也依然花费很多时间访问远端的数据库。

这个并不满足我们“近在咫尺”的愿望。

架构问题

为什么会这样呢？因为有些分布式数据库会有一个限制条件，就是所有的 Leader 节点必须固定在同城主机房，而这就导致了资源使用率大幅下降。TiDB 和 OceanBase 都是这种情况。

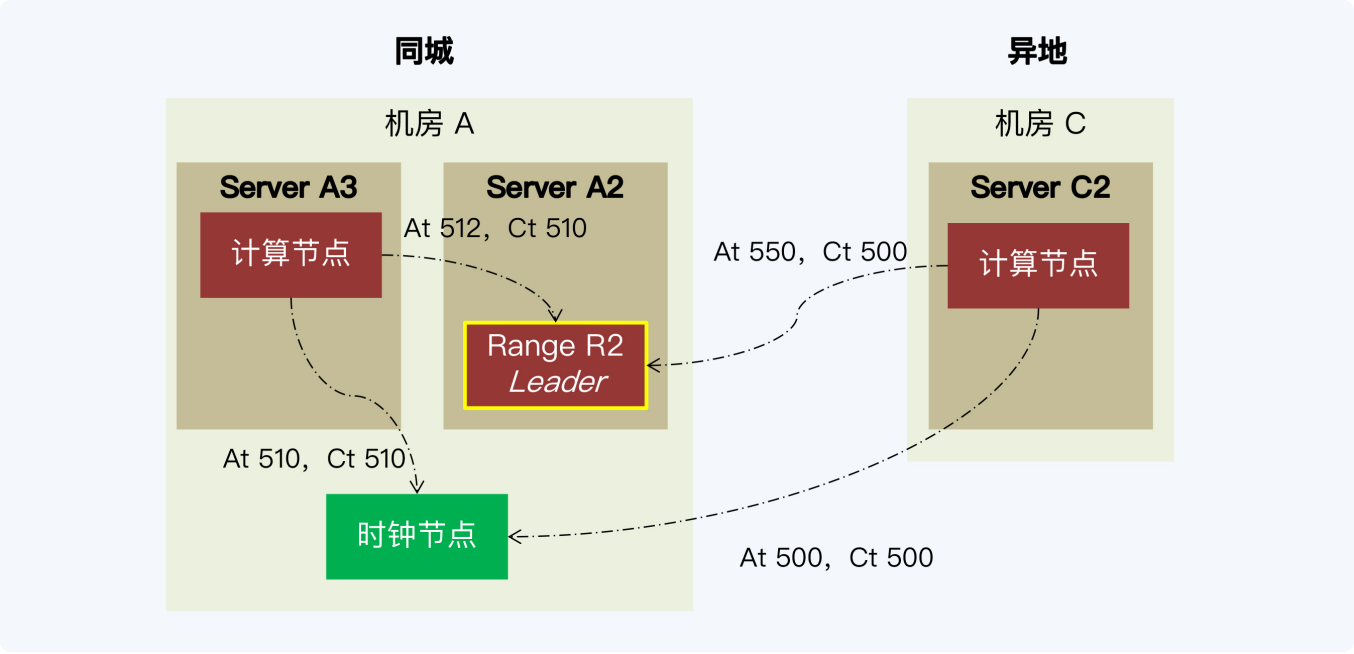
Raft 协议下，所有读写都是发送到 Leader 节点，Follower 节点是没有太大负载的。Raft 协议的复制单位是分片级的，所以理论上一个节点可以既是一些分片的 Leader，又是另一些分片的 Follower。也就是说，通过 Leader 和 Follower 混合部署可以充分利用硬件资源。

但是如果主副本只能存在同一个机房，那就意味着另外三个机房的节点，也就是有整个集群五分之三的资源，在绝大多数时候都处于低负载状态。这显然是不经济的。

那你肯定会问，这个限制条件是怎么来的，一定要有吗？

其实，这个限制条件就是我们在第 5 讲说过的全局时钟导致的。具体来说，就是单时间源的授时服务器不能距离 Leader 太远，否则会增加通讯延迟，性能就受到很大影响，极端情况下还会出现异常。

增加延迟比较好理解，因为距离远了嘛。那异常是怎么回事呢？下面我们就来解释下。



我们把这种异常称为“远端写入时间戳异常”，它的发生过程是这样的：

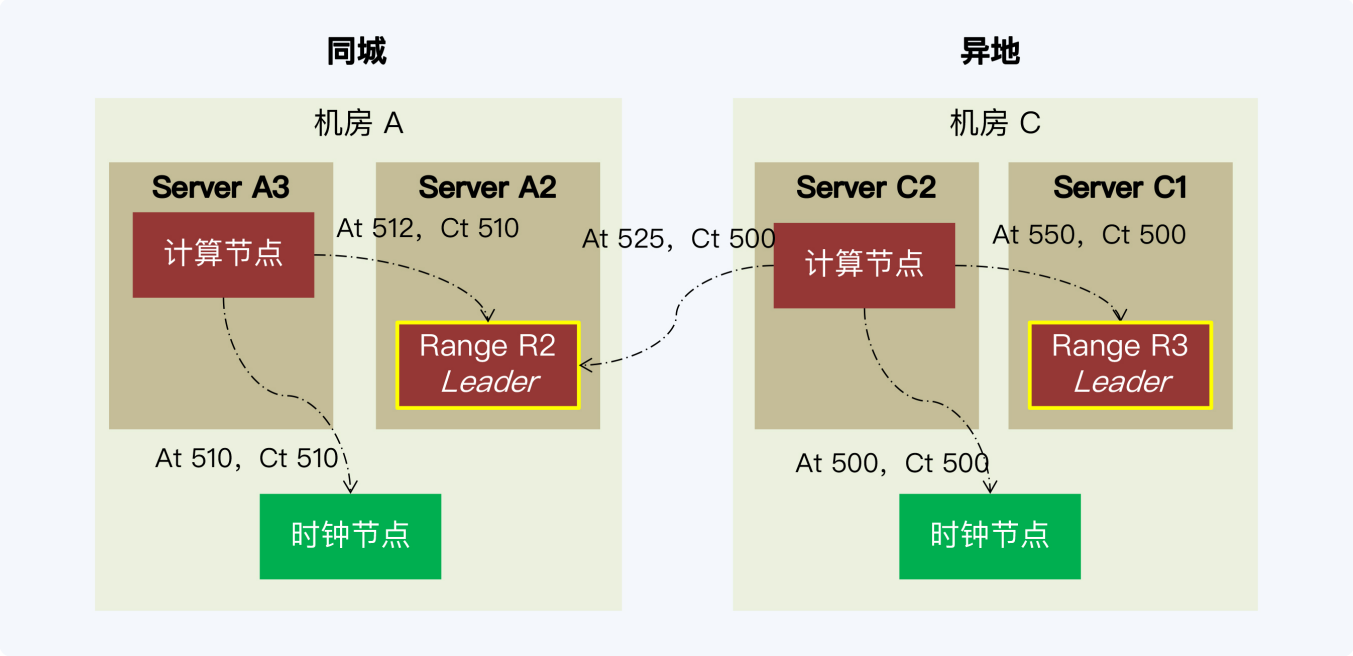
1. C2 节点与机房 A 的全局时钟服务器通讯，获取时间。此时绝对时间（At）是 500，而全局时钟（Ct）也是 500。
2. A3 节点也与全局时钟通讯，获取时间。A3 的请求晚于 C2，拿到的全局时钟是 510，此时的绝对时钟也是 510。
3. A3 节点要向 R2 写入数据，这个动作肯定是晚于取全局时钟的操作，所以绝对时间来到了 512，但是 A3 使用的时间戳仍然是 510。写入成功。
4. 轮到 C2 节点向 R2 写入数据，由于 C2 在异地，通讯的时间更长，所以虽然 C2 先开始写入动作的流程，但却落后于 A3 将写入命令发送给 R2，此时绝对时间来到了 550，而 C2 使用的时间戳是 500。A3 与 C2 都要向 R2 写入数据，并且是相同的 Key，数据要相互覆盖的。这时候问题来了，R2 中已经有了一条记录时间戳是 510，已经提交成功，稍后又收到了一条时间戳是 500 的记录，这是 R2 只能拒绝 500 的这条记录。因为后写入的数据使用更早的时间戳，整个时间线就会乱掉，否则读取的进程会先看到 510 的数据，再看到 500 的数据，数据一致性显然有问题。

这个例子说明，如果远端计算节点距离时钟节点过远，那么当并发较大且事务冲突较多时，异地机房就会出现频繁的写入失败。这种业务场景并不罕见，当我们网购付款时就会出现多个事务在短时间内竞争修改商户的账户余额的情况。

全球化部署

全球化部署的前提是多时间源、多点授时，这样不同分片的主副本就可以分散在多个机房。那么数据库服务可以尽量靠近用户，而应用系统也可以访问本地的分片主副本，整体效果达到等同于单元化部署的效果。

当出现跨多地分片参与同一个分布式事务的情况，全球化部署模式也可以很好地支持。由于参与分片跨越更大的物理范围，所以延迟就受到影响，这一点是无法避免的。还有刚刚提到的“远端写入时间戳异常”，因为每个机房都可以就近获得时钟，那么发生异常的概率也会大幅下降。

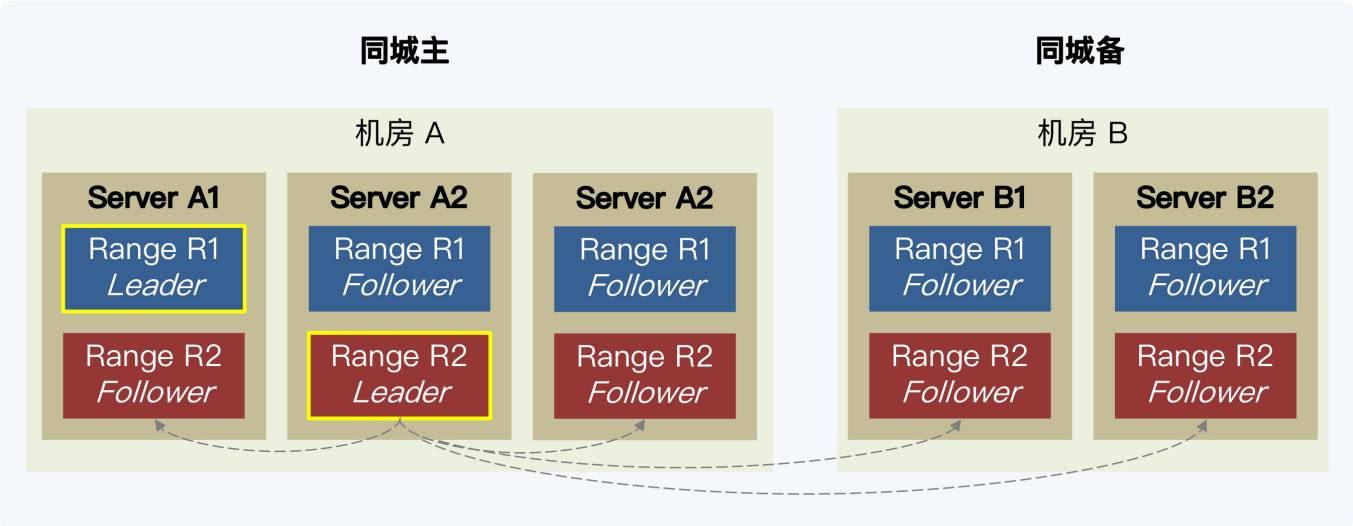


全球化部署模式下，异地机房所有的节点都是处于运行状态的，异地机房不再是替补角色，和同城机房一样提供对等的支持能力，所有机房同等重要。

在任何机房发生灾难时，主副本会漂移到其他机房，整个系统处于较为稳定的高可用状态。而应用系统通过访问本地机房的数据库分片主副本就可以完成多数操作，这些发生在距离用户最近的机房，所以延迟可以控制到很低。这样就做到了我们说的永不宕机和近在咫尺。

同城双机房

接下来，我们再谈一些例外情况，比如下面这种架构。



有了前面的铺垫，再看这种方式你可能觉得有点奇怪。主机房保留了过半的副本，这意味着即使是同城备用机房，也不能实现 RPO 为零。那么主备机房之间就退化成了异步复制，这不更像一个单体数据库的主备模式吗？这样部署的意图是什么呢？

有的运维同学给我解释了这个部署架构，这样可以保证其他机房不存在时，主机房能够单独工作。后来我发现持这种观点的并不是极少数。还有同学会提出来，当主机房只有少数副本时，是不是可以继续工作呢？如果对 Raft 协议有所了解，你会觉得这个要求不可思议，少数副本还继续工作就意味着可能出现脑裂，这怎么可以呢？这里的脑裂是指发生网络分区时，两个少数节点群仍可以保持内部的通讯，然后各自选出的 Leader，分别对外提供服务。

但是换个角度去想，你就会发现这些同学也有他的理由。既然我的数据还是完整的，为什么我不能提供服务呢？虽然，我损失掉了备份机房，但这不影响主机房的工作呀。

如果这个理由也成立，那么哪种选择是更优呢？别急，做判断前，我想和你分享一下对恶意攻击的理解。

恶意攻击

通常来说，架构设计的高可用都是面对正常情况的，机器、网络的不可用都是源于设备自身故障不是外力损坏。但是，有没有可能发生恶意攻击呢？

回到两地三中心的模式，如果三个机房之间的光纤网络被挖断，整个数据库就处于不可用状态，因为这时已经不可能有过半数的节点参与投票了。自然状态下，这个事情发生的概率太低了，三中心之间同时有三条路线，甚至有些机构为了提高安全性，会设置并行的多条线路。但是，如果真的是恶意攻击，多搞几台挖掘机就能让银行的系统瘫痪掉，这比黑客攻击容易多了，而且成本也很低的。

同城双机房的设计，其实一种比较保守的方案，它力图规避了主机房之外因素的干扰因素。为了系统平稳运行，甚至可以放弃 RPO 为零这个重要的目标。虽然我并不认为这是最优的方案，但也确实可以引发一些思考。

RPO 为零是一种保障手段，而持续服务才是目标。那么，也就不应该为了追求 RPO 为零这个手段，而让原本还能正常运行的服务终止掉。这是为了手段而放弃了目标，不就成了舍本求末吗？在必要的时候，还是要保证主要目标舍弃次要的东西。

所以，Raft 协议还需要一个降级机制，也就是说不一定要过半投票，仍然维持服务。类似这样的设计在有些分布式数据库中已经可以看到了。因此，我觉得三地五副本模式加上 Raft 降级，应该算是目前比较完善的方案了。

Follower Read

全球化部署还有一个远端读的性能问题。如果分片的主副本在主机房，而异地机房要读取这些数据，如何高效实现呢？读写分离当然可以，但这损失了数据一致性，在有的场景下是不能接受的。CockroachDB 目前支持的 Follower Read 虽然提升了性能，但也是不保证数据一致性的；而 TiDB 的 Follower 目前还不支持跨机房部署。

但是，这并不妨碍我们探讨未来的可能性，在第 22 讲我提出了一个思路就是利用 Raft 协议无“日志空洞”的特点，等到日志时间戳超过查询时间戳，数据就足够新了。但这仅限于写入操作密集的分片，如果分片上的数据比较冷，根本等不到时间戳增长，又该怎么办呢？

其实还有可优化的地方，那就是利用 Raft 协议的合并发送机制。事实上，在真正实现 Raft 协议时，因为每个 Raft 组单独通讯的成本太高，通常会将同一节点的多个 Raft 协议打包后对外发送，这样可以考虑增加其他分片的最后更新时间戳，再通过协议包的发送时间戳来判断包内分片的最新状态。由于节点级别的 Raft 协议是源源不断发送的，这样只要冷分片和热分片在同一个包内，就可以及时得到它的状态。

全服务的意义

异地多活的终极目标应该是让异地机房提供全服务，也就是读写服务。这样的意义在于让备用机房的设备处于全面运行的状态，这不仅提升了资源利用率，也时刻确保了各种设备处于可运行的状态，它们的健康状态是实时可知的。

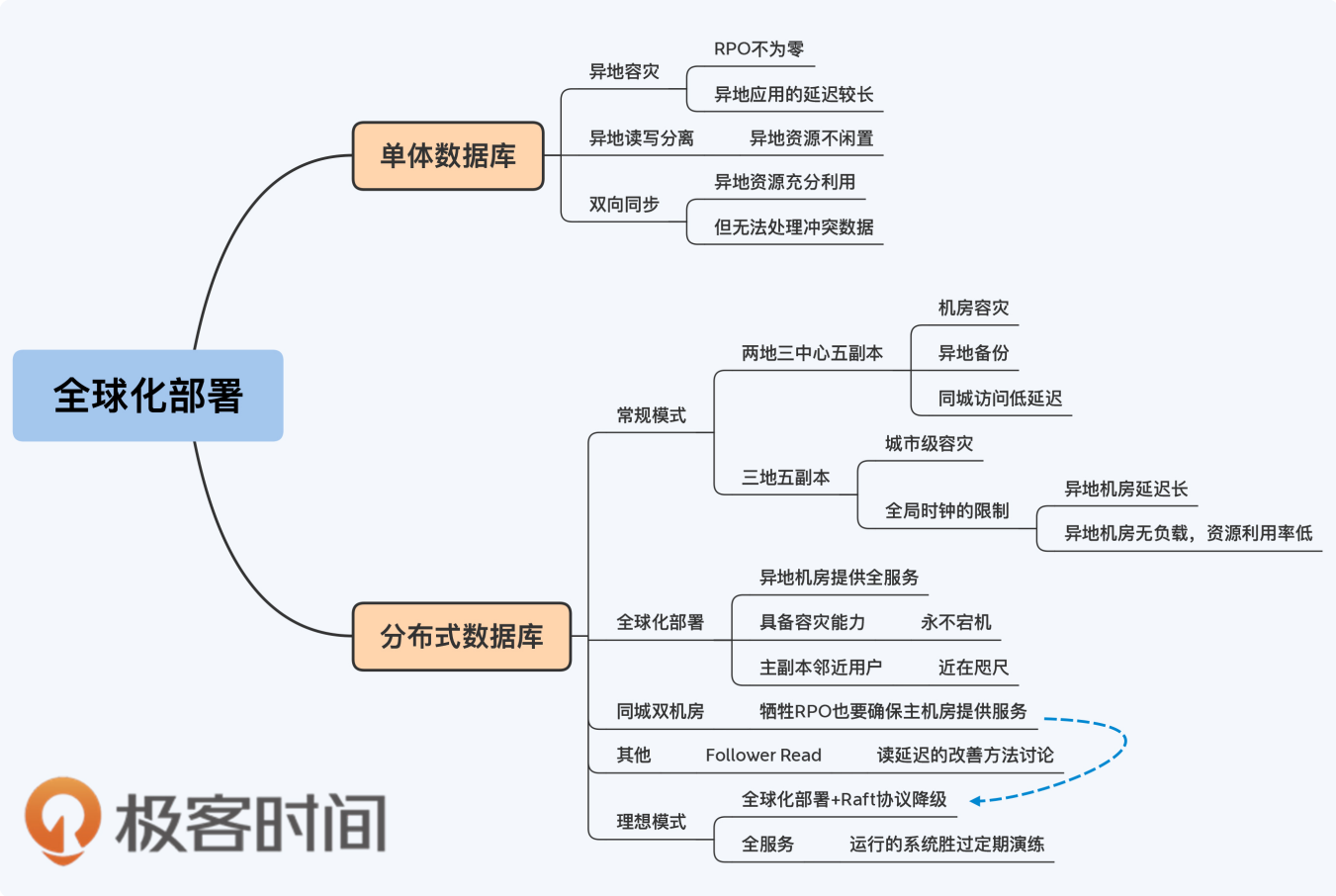
而在异地容灾模式下，备机房必须通过定期演练来确认是可用的，这耗费了人力物力，但并没有转化为真正的生产力，而且仍然存在风险。演练的业务场景足够多吗？出现问题时，这个定期演练的系统真的能够顶上去吗？我想很多人心中或许也是一个问号。显然，一个时刻运行着的系统比三个月才演练一次的系统更让人放心。

所以，我认为一个具有全球化部署能力，或者说是能真正做到异地多活的分布式数据库，是有非常重要的意义的。

小结

那么，今天的课程就到这里了，让我们梳理一下这一讲的要点。

1. 全球化部署就是全球范围下的异地多活。异地多活通常是指系统级别，包括了应用和数据库，难点在于数据库的多活。
2. 单体数据库的异地多活主要有三个版本，异地容灾、异地读写分离和双向同步。但是都无法让应用在同城和异地同时操作相同的数据，没有解决数据库大范围部署的问题。
3. 分布式数据库常采用的部署方式是两地三中心五副本，可以实现机房级别的容灾和异地备份数据但 RPO 不为零。在此基础上，还可以升级到三地三中心五副本，提供城市级别容灾，在邻近城市实现 RPO 为零。使用单点授时的分布式数据库，必须将所有分片的主副本集中在主机房，这一方面是由于访问全局时钟的通讯成本高，另外是为了避免异常现象。
4. 如果从恶意攻击的角度看，基于 Raft 协议的多中心部署反而会带来数据库的脆弱性，因为机房间的通讯链路将成为致命的弱点。所以，应在一定条件下允许对 Raft 协议做降级处理，保证少数副本也可以对外提供服务。
5. 缩短从节点读操作延迟对于异地多活也有重要的意义，目前尚没有完善的解决方案，我们探讨了一些优化的可能性。真正的异地多活必须是异地机房提供全服务，这样才能在本质上提升系统可用性，比定期演练更加可靠。



思考题

课程的最后，我们来看看今天的思考题。今天的课程中，我们提到了 Raft 协议降级处理，它允许数据库在仅保留少数副本的情况下，仍然可以继续对外提供服务。这和标准 Raft 显然是不同的，你觉得应该如何设计这种降级机制呢？

欢迎你在评论区留言和我一起讨论，我会在答疑篇和你继续讨论这个问题。如果你身边的朋友也对全球化部署或者异地多活这个话题感兴趣，你也可以把今天这一讲分享给他，我们一起讨论。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 23 | 数据库查询串讲：重难点回顾+思考题答疑+知识全景图

下一篇 25 | 容灾与备份：如何设计逃生通道保证业务连续性？

精选留言 (2)

写留言

piboye
2020-10-02

raft的降级是要人工运维干预吧？灾难恢复后的数据恢复，要根据业务来处理，写和删除冲突选择写，礼物和金币这类资产类的优先损失公司的，没办法抉择的供用户选择，如果遇到实在没办法恢复的，业务发礼品补偿。

展开

piboye
2020-10-02

真正的全球架构，比异地多活更难吧，美国和中国的时间延迟比上海和深圳的延迟大了10倍吧？